Industrial Strength Pipes

a pipeline toolkit for image processing

Jim Garlick

garlick1@llnl.gov

Center for Application Development and Software Engineering

Lawrence Livermore National Laboratory

Outline

- Introduction
- Application Programming Interface
- Data-Parallel Execution
- Example: SuperMACHO PhotPipe
- Status and Future Work

Introduction

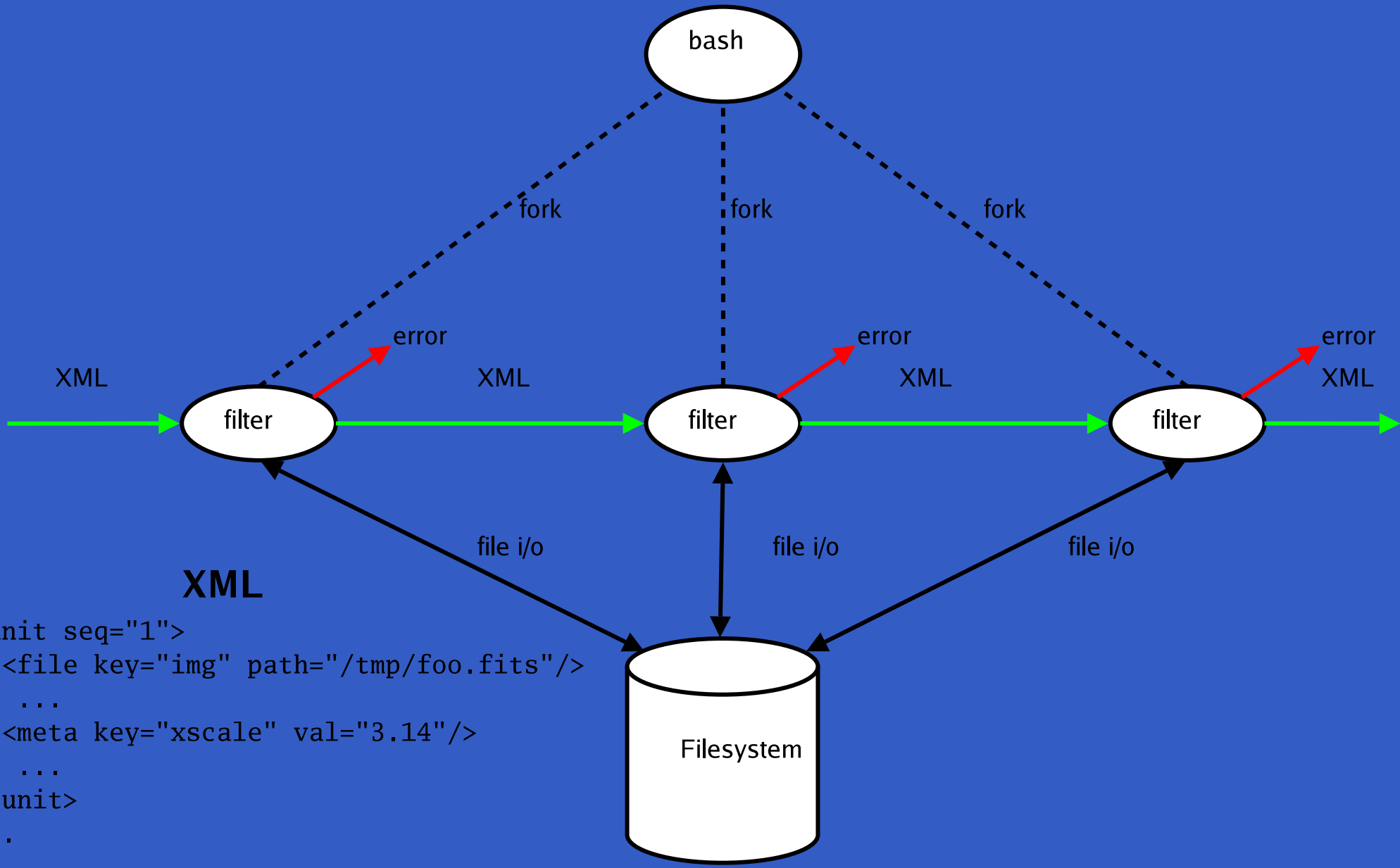
Introduction

- Purpose: Provide API for passing image processing data between filters and a scheme for executing pipelines on large machines.
- Goal: Easy to learn and use.
- Goal: Give users data-parallel execution "for free".
- Goal: Enable further middleware research.
- ISP was influenced by MapReduce from Google and GridDB from UC Berkeley

Overview

- Filters are linked to libisp.a, include isp.h.
- The shell and UNIX pipes tie filters together.
- Pipes carry XML stream of “work units”.
- Work units contain files (passed by reference).
- Work units contain metadata (passed by value).
- Filters produce, consume, or pass through data.

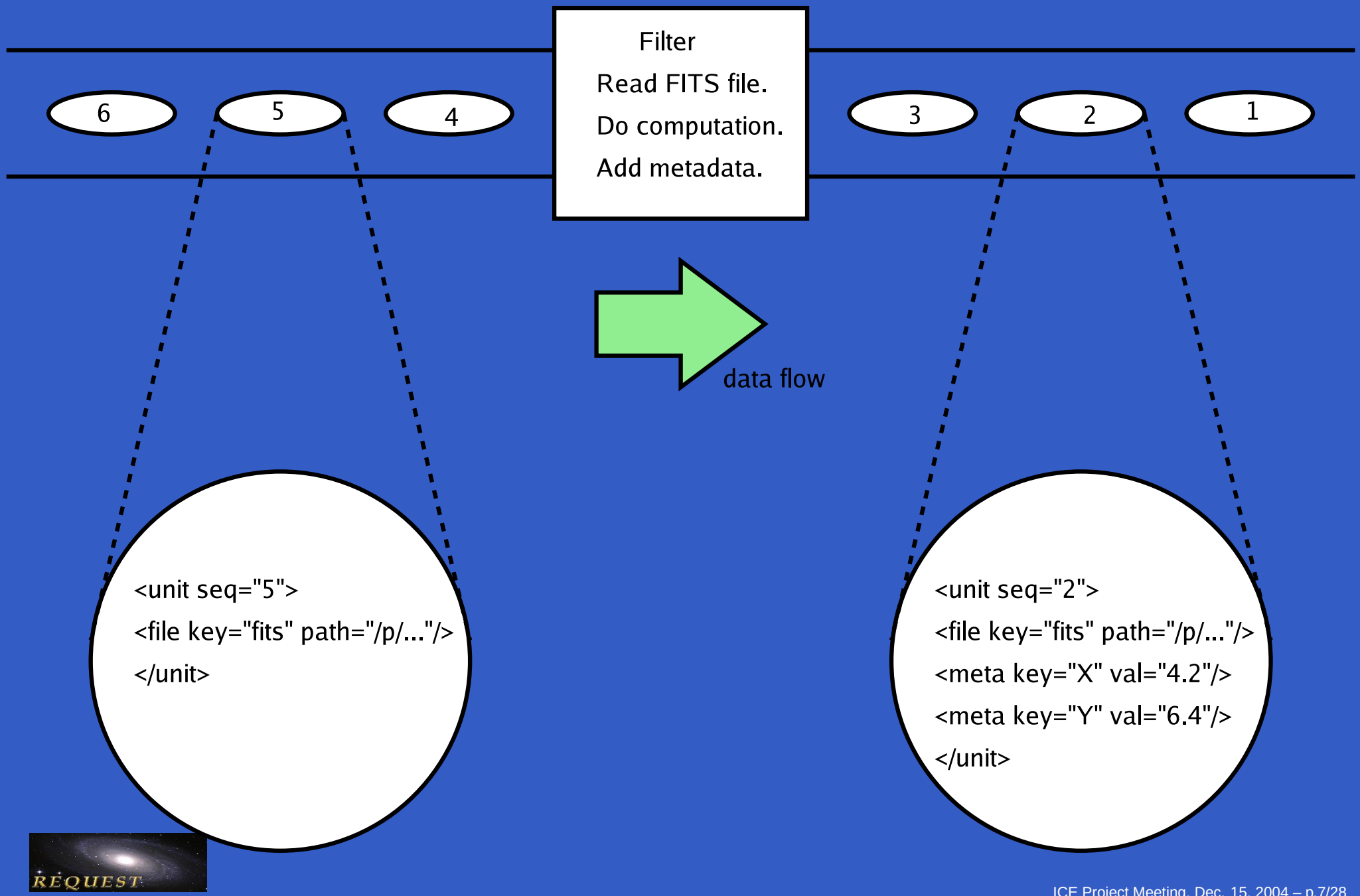
ISP Architecture



XML

```
<unit seq="1">
  <file key="img" path="/tmp/foo.fits"/>
   ...
  <meta key="xscale" val="3.14"/>
   ...
</unit>
...
```

ISP Filters Process “Work Units”



Application Programming Interface

Filter Mainline Interface

Initialization and Finalization	
void	isp_init(int flags, int argc, char *argv[]);
void	isp_fi ni(void);
Unit Creation and Destruction	
isp_unit_t	isp_unit_create(int seq);
void	isp_unit_destroy(isp_unit_t u);
Unit Read and Write	
isp_unit_t	isp_unit_read(void);
void	isp_unit_write(isp_unit_t u);
Unit Map	
typedef int (*isp_mapfun_t)(isp_unit_t u, void *arg);	
void	isp_unit_map(isp_map_fun_t mapfun, void *arg);

Filter Mainline Example

```
#include <isp.h>

int MyFun(isp_unit_t u, void *arg);

int main(int argc, char *argv[]);
{
        isp_init(ISP_SOURCE | ISP_SINK | ISP_PARALLEL, argc, argv);
        isp_unit_map(MyFun, NULL);
        isp_fi n();
}
```

Map Function Interface

File Manipulation

```
int      isp_file_source(isp_unit_t u, char *key, char *path, int readonly);
char *   isp_file_update(isp_unit_t u, char *key);
char *   isp_file_peek(isp_unit_t u, char *key);
int      isp_file_sink(isp_unit_t u, char *key);
int      isp_file_rename(isp_unit_t u, char *key, char *newpath);
char *   isp_mktmp(int *fd);
```

Metadata Manipulation

```
int      isp_meta_source(isp_unit_t u, char *key, char *val);
int      isp_meta_sourcef(isp_unit_t u, char *key, char *fmt, ...);
int      isp_meta_update(isp_unit_t u, char *key, char *val);
char *   isp_meta_peek(isp_unit_t u, char *key);
int      isp_meta_peekf(isp_unit_t u, char *key, char *fmt, ...);
int      isp_meta_sink(isp_unit_t u, char *key);
```

Map Function Example

```
float GetFITSFloat(char *path, char *name);

int MyFun(isp_unit_t u, void *arg)
{
        char *path = isp_fi le_peek(u, "fi ts");

        if (path == NULL) {
                err("could not fi nd fi ts fi le in input");
                return ISP_RET_FAIL;
        }
        isp_meta_sourcef(u, "X", "%f", GetFITSFloat(path, "X"));
        isp_meta_sourcef(u, "Y", "%f", GetFITSFloat(path, "Y"));

        return ISP_RET_OK;
}
```

Data-Parallel Execution

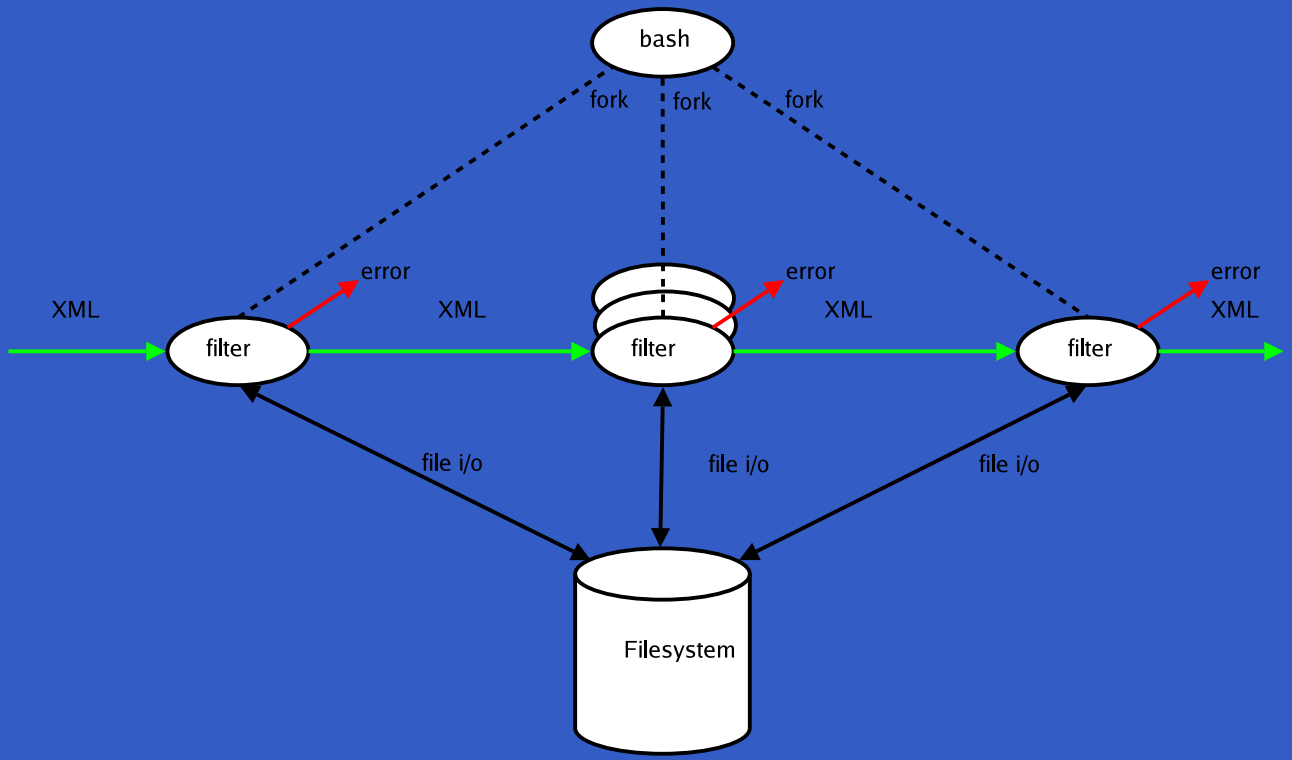
Kinds of Parallelism

- Pipeline parallelism - filters execute concurrently, each processing work units sequentially.
- Data Parallelism - filters process multiple work units concurrently.
- Algorithmic Parallelism - filters execute a parallel algorithm on each work unit.

Data-Parallel Execution

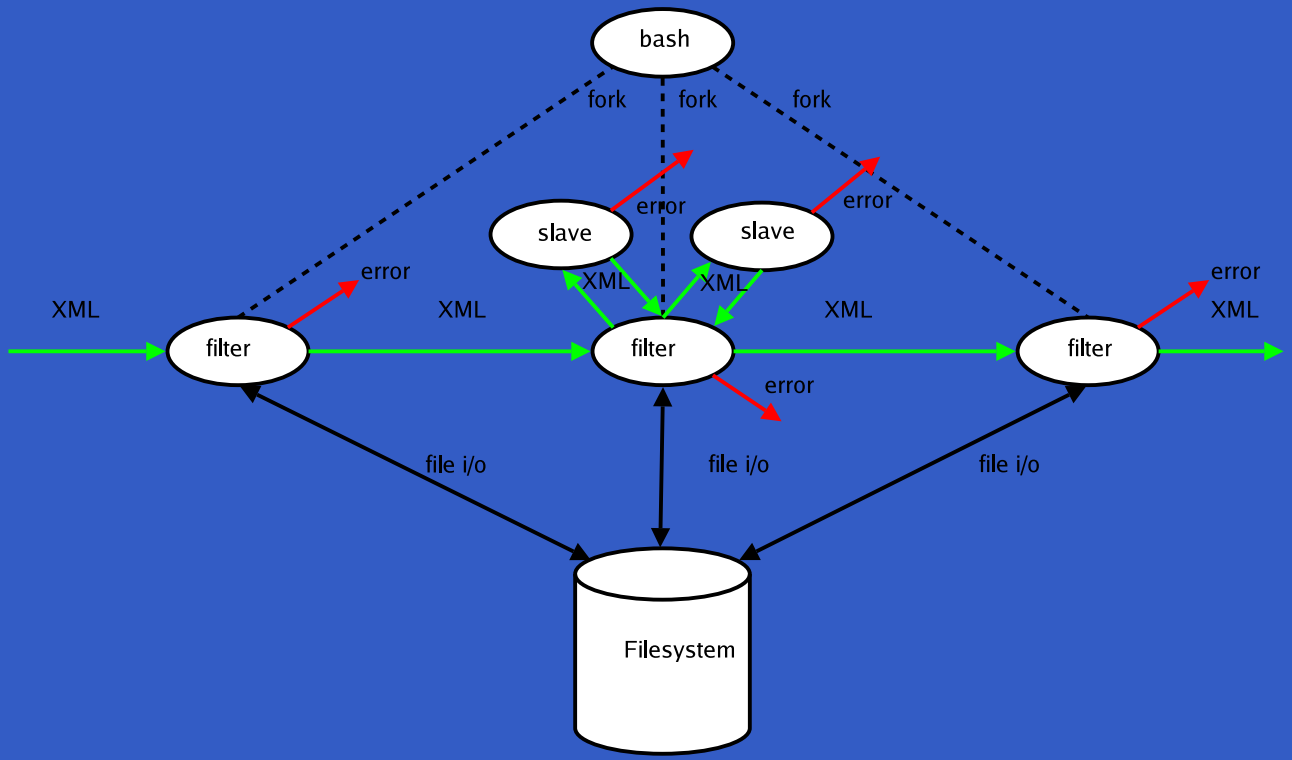
- Map function is executed concurrently on work units.
- Filter requests parallelism via ISP_PARALLEL flag.
- Data-Parallel implementation could use...
  - Pthreads on an SMP.
  - Fork/exec on an SMP.
  - Resource Manager on a distributed memory machine.
- The implementation is hidden from filter programmers.

Data-Parallel Execution with Pthreads



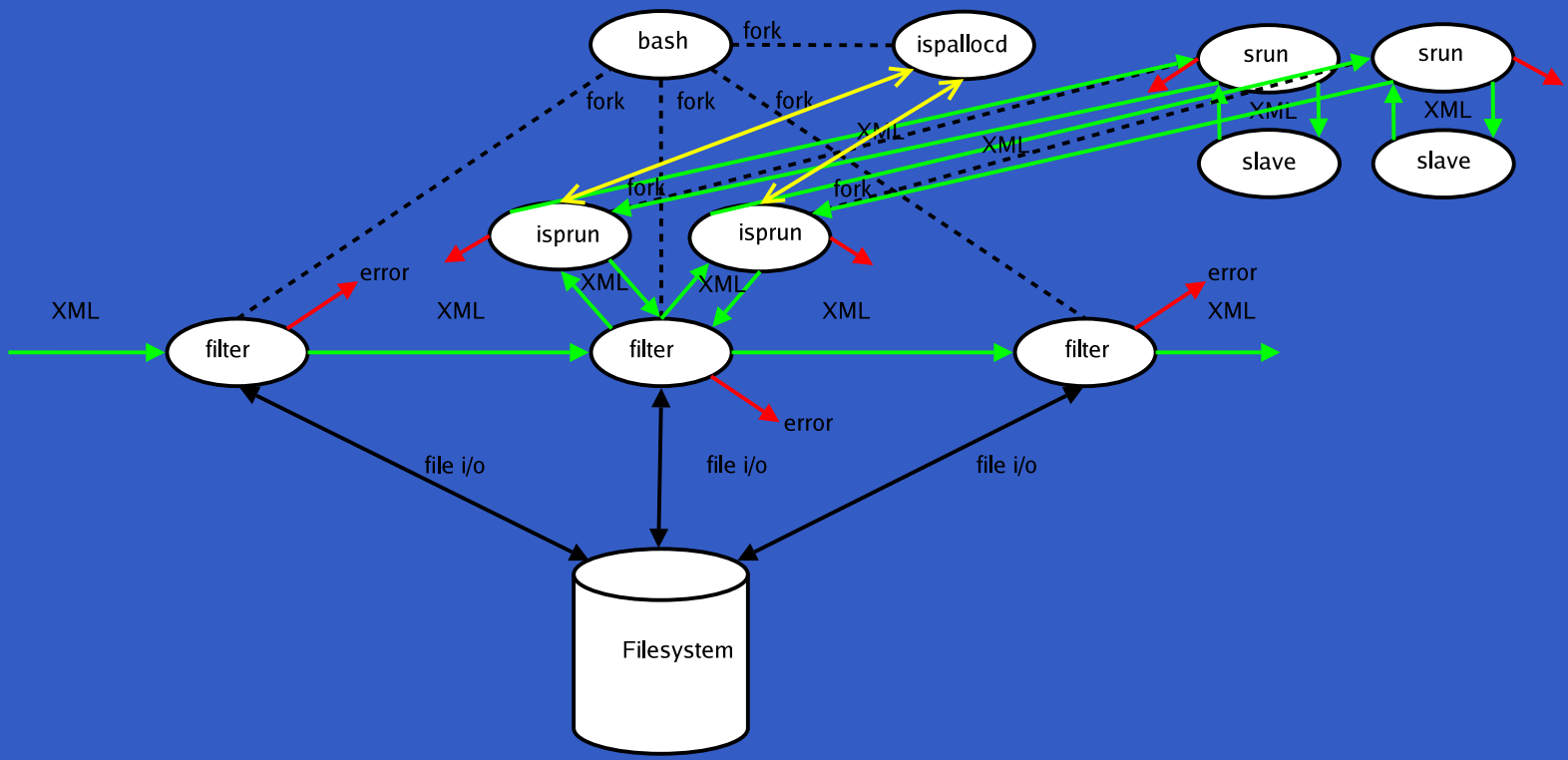
- Main thread manages work crew of threads.
- Mutexes protect XML I/O.

Data-Parallel Execution with Fork/exec



- Parent manages work crew of slave processes.
- Parent must multiplex several I/O streams.

Data-Parallel Execution with SLURM



- Allocator manages requests for CPU/nodes (FIFO).
- Each slave handles one work unit.
- Filter programmers do not need to understand this!

ISP Scripts

- This script can run standalone, under srun (interactive), or under psub (batch).

```
#!/bin/bash
if test $SLURM_NODEID; then
        if test $SLURM_NODEID != 0; then
                exit 0
        fi
        ispallocd -S &
fi
fi lter1 | fi lter2 | fi lter3 >results.xml
```

Example: SuperMACHO Photpipe

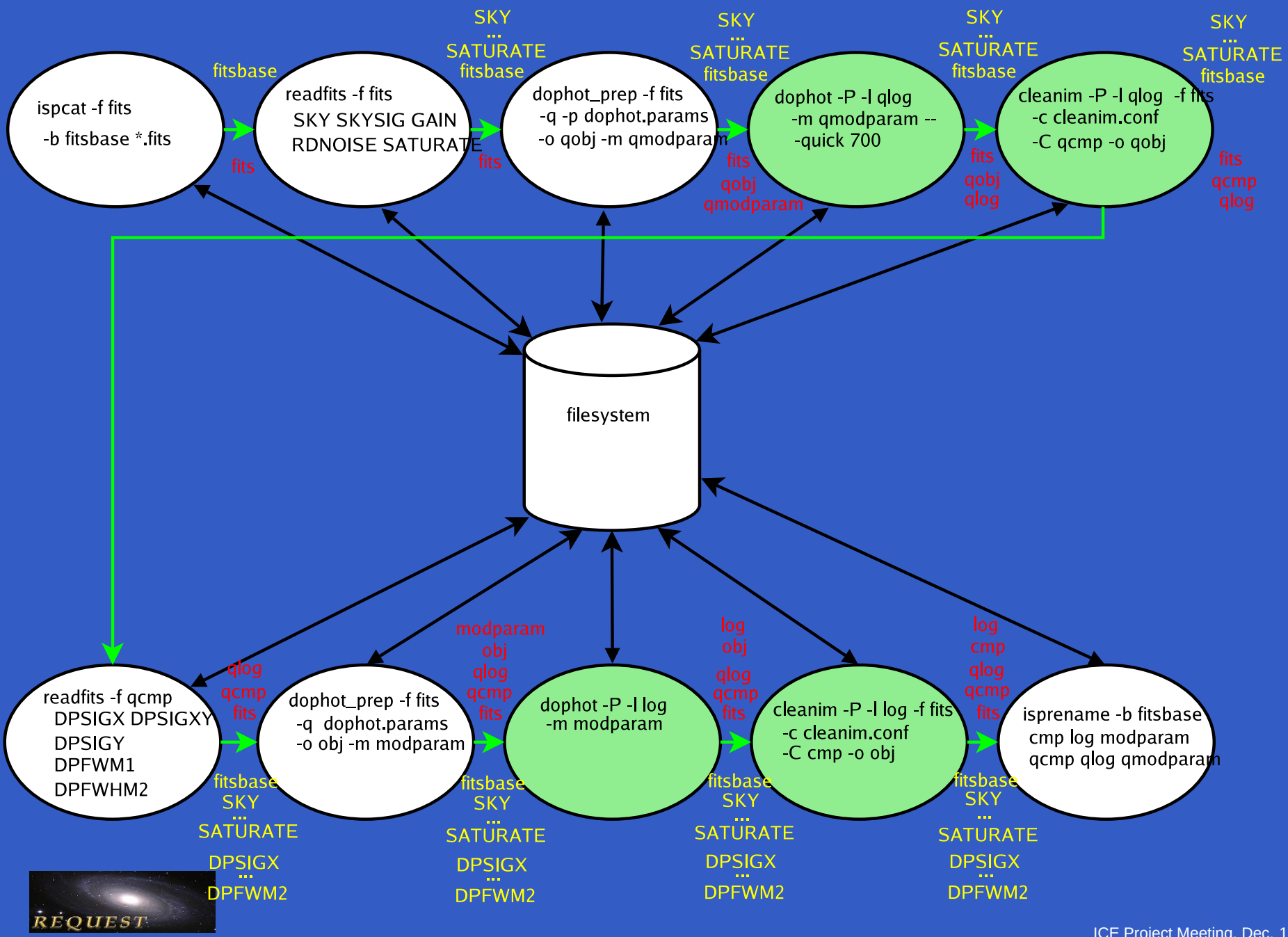
Photpipe Introduction

- Photpipe performs photometry tasks for the SuperMACHO project.
- It converts FITS files $\rightarrow$ CMP files.
- A collection of Perl scripts orchestrates execution.
- Algorithms are separately developed programs, mainly “manna” (image flattening), “cdophot” (source detection), and “cleanim” (file format conversion, etc.).

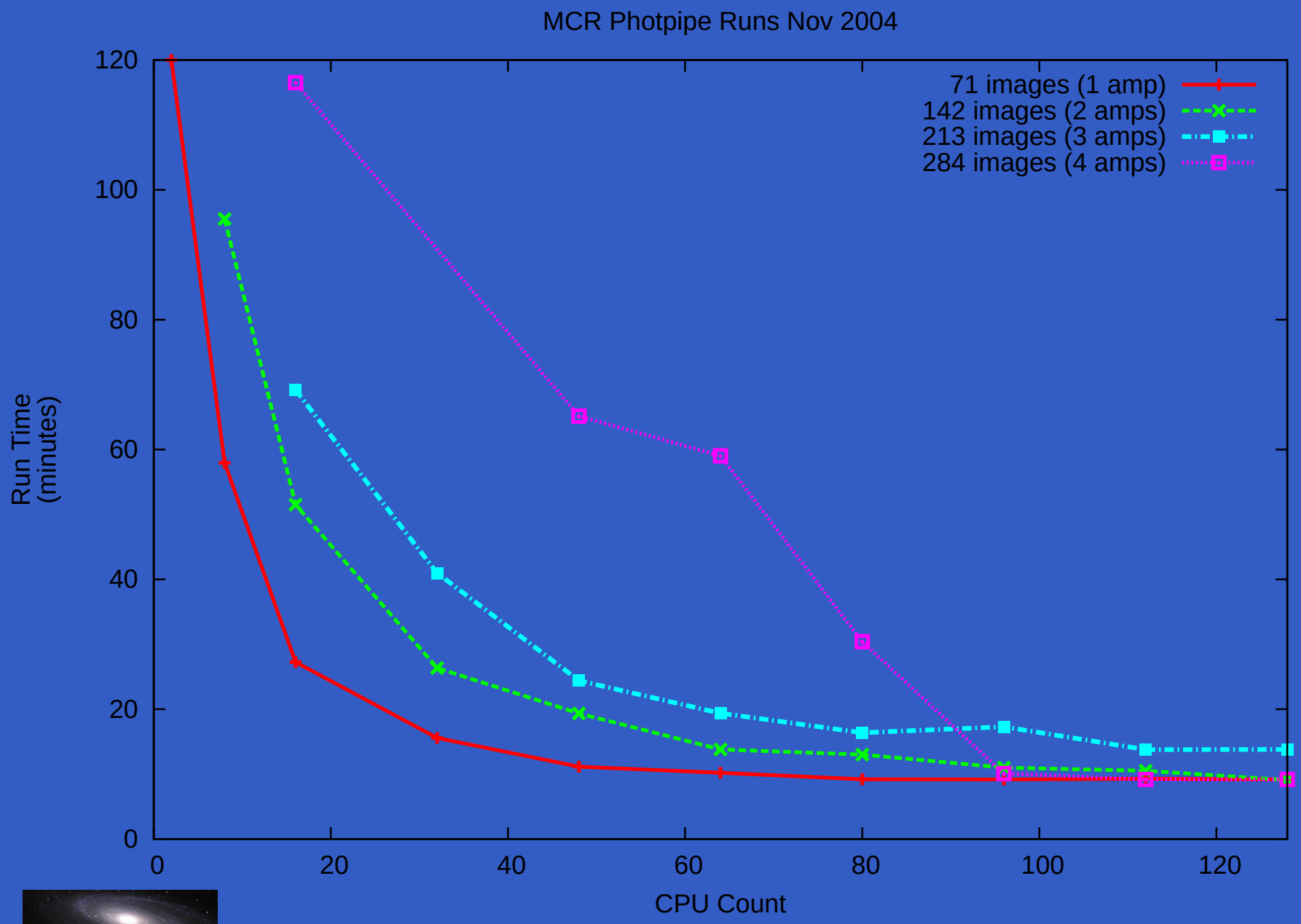
Photpipe with ISP

- Implemented QUICKDOPHOT+CLEANIM; DOPHOT+CLEANIM stages.
- Did not implement FLATTEN stage.
- Some minor algorithmic stuff in perl was converted to additional “stages” in ISP.
- Example: some parameters needed for cdophot were created by reading values from FITS header and performing arithmetic on them.
- This was obfuscated by several layers of nested perl functions in Photpipe.

Photpipe with ISP



MCR Photpipe Scaling Study



Status and Future Work

Version 0.8 Status

- C API bindings are provided.
- Sequential and SLURM data-parallel execution are supported.
- The API needs to be tested against more use cases.
- The filter memory footprint can grow large for long-running pipelines.
- Packaging/cleanup is needed before the first external release.

Future Work

- Debugging tools?
- Checkpoint/restart?
- Logging facility?
- Bindings for Python, Perl, C++, Java?
- More sophisticated pipeline-aware scheduling algorithm?
- Scheduling for multiple pipelines via SLURM plug-in scheduler?
- Use in GridDB for folding/unfolding?

Disclaimer and Auspices

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

UCRL-PRES-209205